AD-A059 794 COMMAND AND CONTROL TECHNICAL CENTER WASHINGTON D C F/G 9/2
NMCS INFORMATION PROCESSING SYSTEM 360 FORMATTED FILE SYSTEM (N--ETC(U)
SEP 78 C K HILL

UNCLASSIFIED CCTC-CSM-UM-15-78-VOL-7 NL

1 OF 1
AD A059794

END
DATE FILMED
12-78
DDC

COMPUTER SYSTEM MANUAL
CSM UM 15-78
VOLUME VII
1 SEPTEMBER 1978

C LEVEL C C T C

COMMAND & CONTROL TECHNICAL CENTER

NMCS INFORMATION PROCESSING SYSTEM 360 FORMATTED FILE SYSTEM (NIPS 360 FFS)

VOLUME VII UTILITY SUPPORT

USERS MANUAL

DEFENSE COMMUNICATIONS AGENCY

(14) CCTC-CSM-UM-15-78-VOL-7

COMMAND AND CONTROL TECHNICAL CENTER

Computer System Manual Number CSM UM 15-78

(6) (11) 1 Sept[illegible]978 (12) 95 p.

NMCS INFORMATION PROCESSING SYSTEM
360 FORMATTED FILE SYSTEM (NIPS 360 FFS).

Users Manual.

Volume III, Utility Support (UT),

(NH)

SUBMITTED BY:

(10) C K Hill

CRAIG K. HILL
~~Captain USN~~
CCTC Project Officer

APPROVED BY:

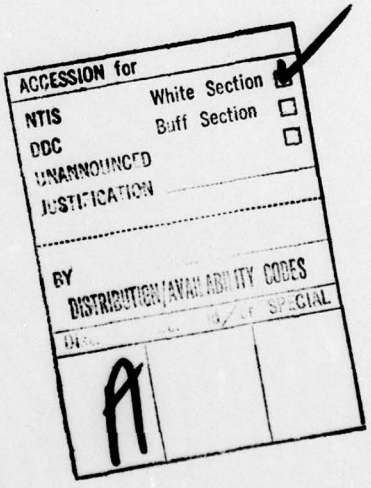
FREDERIC A. GRAP
Captain U. S. Navy
Deputy Director
NMCS ADP

409 658

mt

ACKNOWLEDGMENT

This manual was prepared under the direction of the Chief for Programming with general technical support provided by the International Business Machines Corporation under contracts DCA 100-67-C-0062, DCA 100-69-C-0029, DCA 100-70-C-0031, DCA 100-70-C-0080 DCA 100-71-C-0047 and DCA 100-77-C-0065.



CONTENTS

ABSTRACT

This volume defines the capabilities of NIPS 360 FFS Utility (UT) components. It describes the function of each utility, its inputs, its outputs, and serves as a reference for the knowledgeable user of these components.

This document supersedes CSM UM 15-74, Volume VII.

CSM UM 15-74 Volume VII, is part of the following additional NIPS 360 FFS documentation:

CSM UM 15-78	Vol. I	- Introduction to File Concepts
	Vol. II	- File Structuring (FS)
	Vol. III	- File Maintenance (FM)
	Vol. IV	- Retrieval and Sort Processor (RASP)
	Vol. V	- Output Processor (OP)
	Vol. VI	- Terminal Processing (TP)
	Vol. VIII	- Job Preparation Manual
	Vol. IX	- Error Codes
TR 54-78		- Installation of NIPS 360 FFS
CSM GD 15-78		- General Description

Section 1

INTRODUCTION

This volume of the User Manual contains an analytical description of the general utility support functions provided by NIPS 360 FFS. These functions perform common processing required by all the components. The purpose, use, leck setup, and options of each capability are presented along with clarifying examples.

This volume is divided into the following sections:

a. TABGEN - Discusses the user conversion table generator function

b. SUBLDR - Discusses the user conversion subroutine loader function

c. Data Conversion - Discusses the data base conversion function which converts a 1410 FFS data base to an equivalent NIPS 360 FFS data base

d. File Load/Unload - Provides the Job Control Language to transfer files to and from the Sequential Access Method and the Indexed Sequential Access Method or the Virtual Storage Access Method.

e. UTQRTQDF - Discusses the creation of a NIPS 360 FFS data file from the answer file (QRT/QDF) produced by the retrieval processor (RASP)

f. UTSUBCHK - Discusses the user subroutine checkout function

g. UTDMPLIB - Discusses the capability of printing the names of reports and/or logic statements currently residing on the data file

h. UTCLASS - Discusses the capability of changing the classification of a data file

i. SOURCE LANGUAGE STORAGE - Discusses the library storage of source programs to facilitate housekeeping and program maintenance

j. UTNDXSPC - Discusses the manner in which indexing information may be used without running an FS or FM job

k. UTNDXTFR - Discusses the capability which permits the user to transfer the entire data set, from one resident medium to the other

l. UTFLDSCN - Discusses the source statement field reference scan function.

m. UTNDXKAN - Discusses the capability to analyze the words in fields for which keyword indexing is to be specified.

n. UTNDXKMD - Discusses the maintenance of tables and dictionaries required for keyword indexing.

o. UTODE - Discusses the editing of user format definition source statements and their placement on a user library.

Section 2

TABGEN

TABGEN offers the user the capability to generate data conversion tables that may be used to support input and output functions for NIPS components.

This technique lends itself to effective utilization of file storage space. It also complements the output processor by expanding the coded internal storage values to an external readable form. The NIPS retrieval processors use the reverse technique by permitting the user to express the longer readable form as a search argument. Table conversion will change the search value to the internal storage form for retrieval processing.

Conversion tables consist of argument/function pairs. The argument is called the search value and is the data to be converted. The function is the corresponding converted value. Arguments and functions may be fixed length or variable length.

Tables which are used to convert file data from an internal format to an external format are called Output Conversion Tables. Tables which are used to convert file data from an external format to an internal format are called Input Conversion Tables. Conversion tables may be created to perform both input and output functions. Whatever the function, the table is always entered with the argument and the function is returned. The tables designed by the user and linked to NIPS by the TABGEN processor may be used to:

a. Convert fixed length data to variable length data

b. Convert variable length data to fixed length data

c. Convert fixed length data to fixed length data.

TABGEN may also be used to delete a table or tables from the user's library.

2.1 Input

TABGEN will accept as input the following statements in the order indicated:

a. Delete Table Statement (optional) - Statement is used to request deletion of a table or tables from the user's library.

b. Table Identification Statement (required) - This control statement is used to define the name and function of the table and to describe the format of the table value statements. TABGEN provides the user with two methods of coding control statements. These are the keyword and fixed format methods.

- o Keyword - With the exception of the table name, control parameters may be identified in any order. Each parameter is coded with its associated operands and is keyword identifiable.
- o Fixed - All parameters are column assigned for program interpretation.

c. Header Statements (optional) - Headers are used frequently for title information and will appear on each page of the TABGEN listing.

d. Comment Statements (optional) - Comments are often used to describe the purpose of the table, along with any pertinent remarks the

user wishes to make. These statements will appear only on the first page of the TABGEN listing.

e. Table Value Statements (required) - Table value statements supply the argument/function pairs to the conversion table. The pairs may be arranged for input to the table in either a free or fixed format.

- o Fixed Format - The user is required to align all argument/function pairs in the same column arrangement, but he has the option to choose this alignment as well as the argument/function order. Both arguments and functions may contain embedded blanks.
- o Free Format - The user is not bound to any column alignment constraints for the table value statements; however, a consistent argument/function order must be maintained in the source statements. Either the argument or the function may contain embedded blanks. If a value contains embedded blanks, it must be defined as the variable.

2.1.1 Delete Table Statement

This statement provides the name(s) of tables to be deleted. Any number of cards can be submitted, but they must be first in the TABGEN deck.

The format of the DELETE card is as follows:

DELETE - This keyword which indicates the statement type, must be coded in column 1 of the card.

a. the name of at least one table to be deleted must appear on the card before column 72.

Example: DELETE DTGIS

b. If more than one table name is to be included on the card, they must be enclosed in parenthesis and separated by one blank or comma.

Example: DELETE (DTGIS,DTGOS)

c. A table name or a list of table names cannot be split between cards, but multiple DELETE cards can be submitted.

2.1.2 Table Identification Statement

This statement provides control information to the TABGEN program. The parameters may be keyword or fixed format.

2.1.2.1 Keyword Table Identification Statement

The keyword statement is the method used to express control parameters in free form. With the exception of the conversion table name, the user may specify the necessary control parameters in any order. Only the table name is required in the table identification statement. The remaining parameters are optional and need only be stated if other than default values are required. The following coding conventions apply to keyword table identification statements:

a. Keywords are separated from associated operands with equal signs.

b. Keyword/operand parameters may be separated from other control parameters by a comma or blank(s).

c. Multiple operands assigned to a single keyword will be enclosed in parentheses.

d. More than one card may be used to identify keyword parameters. Code a nonblank character in card

column 72 to indicate continuation of table identification control statements.

e. Do not code parameters beyond card column 71.

f. Do not split keyword/operand lists between card boundaries.

The following statement exemplifies the keyword format:

```
TABLE=OCTEST     ARG=(1/1,ONE,F)          FUNC=(3/19,TWO,V)      X
     USE=0    PAGE=2K     USERNAME
```

The following keyword definitions apply to the keyword statement:

TABLE - This keyword, which specifies the table name, must be coded in column 1 of the table identification card. Table names must conform to standard MIPS name rules as defined in Volume I, Introduction to File Concepts. This is the only keyword that is always required.

Example: TABLE=OCTEST

ARG - This optional keyword may appear anywhere following the TABLE keyword. This defines the characteristics of the table argument. The following subparameters may be coded:

a. hh/ll - This subparameter defines the high- and low-order columns of the argument in the table value statement. Since this defines specific column alignment, the table value cards are, therefore, fixed format.

Example: ARG=(1/1)

b. ll - One to three digits may be used to define the maximum length of the argument. If argument length is specified rather than high- and low-order columns, then the table value cards are free format.

Example: ARG=1

If neither argument length nor column alignment is specified, the table value cards will be considered free format. The actual argument length will be determined from the maximum length of the arguments in the table. If argument length is greater than 30 characters, the maximum length must be entered in the PARM field of the EXEC JCL card. The user may not specify both column alignment and length on a table identification statement.

c. ONE - The subparameter 'ONE' or 'TWO' may be coded to indicate the relative position of the argument to the function in the table value card. 'ONE' is the default value for the argument; 'TWO' is the default value for the function. However, if the user overrides either position subparameter, the remaining subparameter automatically "flip-flops" to an opposite default value. If the position specification is not consistent with column alignment requirements for fixed format table value cards, the column alignment specifications take precedence.

Examples: ARG=(1/1,ONE)

Specifies fixed format table value card, with argument preceding function.

ARG=(5,TWO)

Specifies free format table value card. Argument is a maximum of five bytes long and follows the function on the table value card.

d. F - The subparameter 'F' indicates the argument is fixed in length (no embedded blanks). 'V' indicates the argument is variable in length. If this subparameter is omitted, the default for the argument is 'F'.

Example: ARG=(1/1,ONE,F)

FUNC - This optional keyword with its subparameters may appear anywhere on the table identification card following the TABLE keyword. This defines the characteristics of the table function. The following subparameters may be coded:

a. hh/ll - This subparameter defines the high- and low-order columns of the function in the table value statement. Since this defines specified column alignment, the table value cards must be fixed format.

Example: FUNC=3/19

b. lll - One to three digits may be used to define the maximum lengths of the function. If the function length is specified rather than high- and low-order columns, then the table value card must be free format.

Example: FUNC=17

If neither function length nor column alignment is specified, the table value cards will be considered free format. The actual function length will be determined from the maximum length of the functions in the table. This may not exceed 255 bytes. The user may not specify both column alignment and length on a table identification statement.

c. TWO - The subparameter 'TWO' or 'ONE' may be coded to indicate the relative position of the function to the argument in the table value card. 'TWO' is the default value for the function; 'ONE' is the default value for the argument. However, if the user overrides either position subparameter, the remaining subparameter automatically "flip-flops" to an opposite default value. If the position specification is not consistent with column alignment requirements for fixed format table value cards, the column alignment specifications take precedence.

Examples: FUNC=(3/19,TWO,V)

This specifies a fixed format table value card with the function in columns 3-19. The function is variable length and follows the argument on the table value card.

FUNC=(15,ONE)

This specifies a free format table value card with a maximum function length of 15 bytes. The function defaults to variable length and precedes the argument on the table value card.

d. V - The subparameter 'V' indicates the function is variable in length. 'F' is the user's option and indicates the function is fixed length. If this function subparameter is not coded, the default value is 'V.'

USE - This keyword parameter may appear anywhere on the table identification statement card following the TABLE keyword. The parameter indicates the use of the table for:

I = Input
O = Output
B = Both

If the keyword is not coded, 'B' is assumed.

PAGE - This keyword may appear anywhere following the TABLE keyword. The subparameter indicates the maximum size of a page for the table. The user has the option to specify the following page sizes:

1K
2K
4K

If this keyword does not appear on the table identification statement, a page size of 1K is assumed.

USERNAME - The user's name may appear anywhere on the table identification card after the TABLE keyword. This parameter is optional.

2.1.2.2 Fixed Format Table Identification Statement

This statement is used to express control parameters in fixed format. As in the case of the keyword table identification statement the table name and function, as well as the format of the table value statements, will be defined in this statement. Specific column alignment of parameter values is essential for program interpretation. The following card column alignment defines input formats for both fixed and free format table value statements.

Card Columns	Fixed Format Table Value Statements	Free Format Table Value Statements
1-5	Contains card identifier	Contains card identifier
	TABLE	TABLE
11-17	Contains name to be given to a fixed-to-variable-length conversion table	Same
21-27	Contains name to be given to a variable-to-fixed-length conversion table	Same
31-32	High-order position of fixed length field in the table value statement	Length of fixed length field in the table value statement
34-35	Low-order position of Fixed length field in the table value statement	Blank

Card Columns	Fixed Format Table Value Statements	Free Format Table Value Statements
41-42	High-order position of variable length field in the table value statement	Length of variable length field in the table value statement
44-45	Low-order position of variable length field in the table value statement	Blank
50	Contains code designating use of table: I = Input O = Output B = Both	Same
51	Blank	Contains 'F' if the fixed field is the first field in the table value statement; contains 'S' if the fixed field is the second field in the table value statement
55	Contains the user's name	Same
73-80	Card deck sequence number if desired	Same

2.1.3 Header Statement

Header statements are optional and may be used with keyword or fixed format control statements. For each table generated by the TABGEN processor, a listing will be provided to the user. Up to four user-specified header lines may be printed on each page of the generated table listing. A header statement is identified by one, two,

three, or four asterisks left-justified in columns 1-4 representing the statements (lines) one, two, three, or four. Data printed on each page of the output listing is taken from columns 10-80 of a statement. For example:

```
Card Column
1         10

*         THIS IS AN OUTPUT HEADER
**        THAT WILL APPEAR IN THIS ORDER
***       ON EACH PAGE OF OUTPUT OF
****      A TABGEN LISTING.
```

2.1.4 Comment Statement

A comment statement is identified by an asterisk in column 6. Any number of comment statements may be applied by the user, but they must follow the header statement and precede the table value statements. The contents of a comment statement will be printed exactly as it appears on the punched card. For example:

```
Card Column
      6  10

      *  THIS IS A COMMENT STATEMENT
      *  THAT WILL APPEAR ON THE FIRST
      *  PAGE ONLY OF A TABGEN LISTING.
      *  THERE MAY BE ANY NUMBER OF
      *  COMMENT STATEMENTS PRIOR TO THE
      *  OUTPUT OF THE TABLE VALUE STATEMENTS.
```

2.1.5 Table Value Statements

Argument/function pairs are supplied to TABGEN by table value statements. If the keyword table identification statement is used, the user may create a table in which the argument/function pair may have a combined length of 256 characters. If this method is used and the argument/function pair exceeds 71 characters, the table value statements should be continued to the next card. A

maximum of four continuation cards is permitted. A non-blank character in column 72 will indicate continuation. The scan will proceed through column 71 and continue with column 1 on the next card.

If the fixed table identification statement is employed, the argument/function pair must be contained within the column boundaries of one punched card.

In either case, table value statements may appear in one of two formats; i.e., fixed or free. Examples of each follow.

a. Fixed Format - The user is required to align all argument/function pairs in a prescribed column arrangement. The column alignment and argument/function order are optional; however, all table value inputs must have the same format. For example, the following table value statements would supply argument/function pairs in fixed format to a "Service Table":

Card Column

1 3 19
E U.S. Coast Guard
W U.S. Army
J U.S. Air Force
N U.S. Navy
M U.S. Marine Corps

Using the fixed format table value statements, the high- and low-order positions of the argument/function pairs must be specified in the table identification statement. Fixed format table value statements can contain blanks in both fixed and variable length fields.

b. Free Format - The user is not limited by column boundary restrictions using the free format method for input of table value statements. Data may start in any card column and end in any card column between columns 1 and 71, inclusive, for keyword

table identification control cards, and between columns 1 and 80, inclusive, for fixed table identification control cards. The order of argument/function pairs must be indicated and remain consistent for all inputs to a single table. For example, the following table value statements would supply argument/function pairs to a "Service Table":

```
E      U.S. Coast Guard
   W      U.S. Army
 J            U.S. Air Force
  N      U.S. Navy
M U.S. Marine Corps
```

The order of argument/function pairs in the example shown remains consistent in each table value statement. However, column selection of data value placement is free format. In this format the fixed length field cannot contain embedded blanks, but the variable length field can.

2.2 Job Setup

a. The following job setup is used to execute the XTABGEN cataloged procedures and must be organized in the order shown:

```
//Jobname        JOB   (Standard parameters)
//Stepname       EXEC XTABGEN,LIB=yourlib
//TAB.SYSIN      DD  *
TABLE IDENTIFICATION STATEMENT (Required)
HEADER STATEMENT                (Optional)
COMMENT STATEMENT              (Optional)
TABLE VALUE STATEMENTS         (Required)
/*
```

b. More than one table may be generated in a single job step. If tables are batched and a table argument length exceeds 30 bytes, TABGEN must be informed by stating the maximum argument length in the PARM field of the EXEC card, for example, PARM=nnn (nnn is any number between 1 and 255).

c. The generated table will become an executable load module which will reside as a member of a library that will be identified by the LIB= parameter in the EXEC card. The user may create his own new library by overriding the following symbolic parameters:

```
LIBDISP=  (NEW,KEEP)
VLIB=     (volume)
LIBSP=    (space)
ULIB=     (unit)
```

2.3 Limitations

a. The maximum table size is approximately 528,000 bytes The maximum number of pages which any table may contain is 132.

b. The maximum argument/function size is:

   o. 256 bytes when using keyword table identification statements

   o. 80 bytes when using fixed table identification statements.

2.4 Examples

Examples of test runs using keyword table identification statements.

a. The following TABGEN source deck setup was used to generate a fixed-to-variable length table using free format table value statements. The table name is CTRYS, and the table converts a two-character internal storage code to an output value with a maximum length of 15 characters. The user name is TABGENTEST and the table page size is 2,000 characters.

```
//Jobname       JOB   (standard parameters)
//STEPA         EXEC  XTABGEN,LIB=yourlib
//TAB.SYSIN     DD  *
TABLE=CTRYS ARG=2 FUNC=15 USE=0 PAGE=2K TABGENTEST
*    TABLE CTRYS USING FREE FORMAT TABLE
**   IDENTIFICATION CARD AND TABLE VALUE
***  CARDS.
AA  AFRICA
    AC          ATLANTIC OCEAN
.....................................................
64        CARIBBEAN
       65 PACIFIC ISLANDS
/*
```

The output listings for this source deck will appear as follows:

```
DATE 71001  TABLE-CTRYS  ORIGINATOR TABGENTEST  PAGE-001

     TABLE CTRYS USING FREE FORMAT TABLE
     IDENTIFICATION CARD AND TABLE VALUE
     CARDS.

ARGUMENT  FUNCTION
--------  --------
   AA       AFRICA
   AC       ATLANTIC OCEAN
  ........................
   64       CARIBBEAN
   65       PACIFIC ISLANDS
```

b. The following TABGEN source deck setup was used to generate a variable-to-fixed length conversion table using fixed format table value statements. The table name is UNLVSI, and the table is used to convert an external value with a maximum length of 15 characters to an input storage code with a maximum length of three characters. The user name is TABGENTEST and the table page size is 1,000 bytes:

```
//Jobname       JOB  (standard parameters)
//STEPA         EXEC XTABGEN,LIB=yourlib
```

```
//TAB.SYSIN     DD  *
TABLE=UNLVSI ARG=10/24 FUNC=1/3 USE=I TABGENTEST
*      TABLE UNLVS USING KEYWORD TABLE IDENTIFICATION
**     STATEMENTS AND FIXED FORMAT TABLE VALUE
***    CARDS.
A      NUMBERED ARMY
ACD    ACADEMY
.....................................................
U      UNIT
USS    US SHIP
WG     WING
/*
```

The output listing for this source deck will appear as follows:

```
DATE 71007  TABLE-UNLVSI  ORIGINATOR-TABGENTEST  PAGE-001

     TABLE UNLVS USING KEYWORD TABLE IDENTIFICATION
     STATEMENT AND FIXED FORMAT TABLE VALUE
     CARDS.

ARGUMENT          FUNCTION
--------          --------
NUMBERED ARMY     A
ACADEMY           ACD
.....................................
UNIT              U
US SHIP           USS
WING              WG
```

Section 3

SUBLDR

SUBLDR is used to transfer a conversion subroutine in load module form from a work library to the NIPS library. This procedure establishes the proper linkage for the interface between the NIPS 360 FPS and the user subroutine. The subroutine should have been tested previously by the user. The conventions required in writing the subroutine have been outlined in Volume I, Introduction to File Concepts.

3.1 Input

Input for the cataloged procedure XSUBLDR comes from two sources: a user-supplied statement in the job deck defining the attributes of his subroutine, and the subroutine in load module form. The location of this load module is identified by a symbolic parameter supplied by the user.

The free format control statement defines the subroutine, and is punched on a card. The parameters must be in their stated order but may be separated by blanks or commas and may start in any card column. The nine parameters used in the control statement are as follows:

a. Statement Identifier - SUBRT

b. Conversion Subroutine Name - Name to be used in NIPS 360 FPS statements when invoking the subroutine. This name must conform to specifications outlined in Volume I, Introduction

to File Concepts, and must be unique in the MIPS library where the subroutine is stored.

c. Maximum Argument Size in Bytes - Decimal number (maximum) to be accepted by the subroutine.

d. Maximum Function Size in Bytes - Decimal number (maximum) to be supplied by the subroutine.

e. Conversion Subroutine Type

I - Input conversion
O - Output conversion
B - Input/output conversion.

f. Argument Mode - For input data to the subroutine.

A - Alphameric mode
B - Binary mode
C - Coordinate mode
D - Decimal mode

g. Function Mode - One of the characters listed in parameter (f) defines the mode of output data from the subroutine.

h. Subroutine Load Module Name - As it exists on a work library, this name must be the same as the load module entry point name. This name may be the same as that used in parameter (b) but not necessarily. The name used may not be greater than seven characters in length.

i. User Name - Writer of the subroutine. Up to 18 characters may be used, with no embedded blanks. Periods may be used to separate initials.

3.2 Job Setup

The following statements illustrate the deck setup used to execute the ISUBLDR procedure.

```
//JOBNAME    JOB    (standard parameters)
//             EXEC   XSUBLDR,LIB=yourlib,MODLIB=TEMP
//SUB.SYSIN  DD     *
  Control statement defining subroutine
  as described in 3.1.
/*
```

The LIB symbolic parameter defines the library where the executable subroutine load module is to be stored. This can be a user's private library or an installation MIPS library.

The MODLIB symbolic parameter provides the work library name where the subroutine load module is located.

This library can be the installation program library or a private library built by the user when the subroutine is assembled, link edited, and tested by the user.

Sample Control Statement

```
SUBRT, CTRYS, 02, 15, 0, A, A, MODNM, USERNAME
```

This control statement defines the MIPS subroutine CTRYS as an output subroutine accepting two alpha argument bytes and generating 15 alpha bytes output. The load module created by the user has name and entry point MODNM.

Section 4

DATA CONVERSION

The data conversion utility allows the user/analyst to convert 1410 FFS data bases to HIPS 360 FFS format. HIPS 360 FFS data bases can be reconverted to 1410 FFS format. In either mode, the conversion process produces a logically identical file containing all of the appropriate elements of information for the mode. The process is essentially automatic, requiring the analyst to provide an FFT for each of the two modes of files, but requesting no control data.

Data conversion occurs at the field level, permitting rearrangement of data elements within a periodic subset or fixed set. New fields may be added and old fields may be dropped. The variable set (VSET) of a 1410 file may not be dropped when converting to a HIPS 360 FFS format. Repositioning of the data fields within the subset or fixed set permits change of the record control group contents. The system senses any variation in the sequence of the new file and prints an error message when the file must be sorted.

The 1410 mode coordinate fields are automatically converted by the system only when the length specification in the 1410 FFT is equal to 13. Since this is the normal internal format for 1410 data bases, no problem should result from this constraint.

A listing of all field names, and their disposition, will be printed at the beginning of the run to aid the analyst in determining new file contents. At the end of the

run, a space allocation print-back is included to assist in loading the data base onto direct access devices. The conversion process is tape-to-tape when going to NIPS 360 format and disk-to-tape when going to the 1410 format.

DD cards used in the utility (UTDATAC) are:

DDNAME	File Described
SYSIN	1410 object FPT
DATAFILE	360 ISAM FPT (and data base if 360 to 1410)
FILE1410	1410 data base (optional, see below)
NEWFILE	Output data base from conversion
SYSPRINT	)
SYSOUT	) Printer

Direction of conversion will be determined by the presence of the FILE1410 DD card; if it is present, the conversion is from the 1410 to the 360.

Description of data sets:

SYSIN - 1410 FPT in object deck form.

DATAFILE - 360 ISAM FPT/data base.

FILE1410 - 1410 data base, usually a 7-track, 556 bpi, even-parity tape with nonstandard labels. The following example is for the SOFAA data base which is cataloged on the 360:

```
//FILE1410 DD DISP=OLD,DSNAME=SOFAAXXX,LABEL=(,NSL),    C
//            DCB=(DEN=1,TRTCH=ET)
```

NEWFILE - Output file from the conversion utility. For the 1410 to 360 conversion, this will describe a SAM file on 9-track tape.

```
//NEWFILE DD UNIT=2400,DISP=(,CATLG),DSNAME=SOFAA
```

For the 360 to 1410 conversion, this will describe a seven-track tape with nonstandard labels. The BLKSIZE must be supplied in the DCB parameters.

```
//NEWFILE  DD UNIT=2400-2,DISP=(,KEEP),DSNAME=SOFAAXXX, C
//           LABEL=(,NSL),DCB=(DEN=1,TRTCH=ET,BLKSIZE=2704)
```

SYSPRINT AND SYSOUT - These describe the system output writers.

4.1 Data Conversion Utility - UTDATAC

This utility will convert a 1410 FFS data base to a NIPS 360 FFS data base. It will reconvert a NIPS 360 FFS data base to a NIPS 1410 FFS data base.

4.1.1 Procedure X360CON

This procedure will convert a 1410 FFS data base to a NIPS 360 FFS data base. The 1410 FFS file is assumed to be on a 7-track tape and the new data base is to be written on a 9-track tape. Using symbolic parameters for this procedure, one would use the following JCL:

```
//JOBNAME     JOB    (standard parameters)
//  EXEC    X360CON,ISAM=aaaaaaa,VISAM='SER=bbbbbb',    X
//          FL1410=ccccc,V1410='SER=dddddd',            X
//          SAM=eeeeeee,VSAM='SER=ffffff'
//SYSIN       DD     *
                  -----
              (FFT object deck of 1410 FFS)
                  ----
/*
```

where

```
aaaaaaa = NIPS 360 FFS ISAM data base name on disk
bbbbbb  = disk serial number for the ISAM data base
ccccc   = 1410 FFS tape data base name
dddddd  = tape serial number for the 1410 data base
eeeeeee = NIPS 360 FFS tape data base name
ffffff  = tape serial number for the 360 data base.
```

4.1.2 Procedure X1410CON

This procedure will convert a NIPS 360 FFS data base to a 1410 FFS data base. The NIPS 360 FFS data base is assumed to be on a 2314 disk pack and the converted data base is written on a 7-track tape with nonstandard labels. Using symbolic parameters for this procedure, one would use the following JCL:

```
//JOBNAME     JOB    (standard parameters)
//       EXEC  X1410CON,ISAM=aaaaaaa,VISAM='SER=bbbbbb',    X
//       SAM=eeeee,VSAM='SER=ffffff'
//SYSIN  DD   *
                    ----
                (1410 FFS FFT object deck)
                    ----
/*
```

where

```
aaaaaaa = NIPS 360 FFS ISAM data base name on disk
bbbbbb  = disk serial number for the ISAM data base
eeeee   = data base name for the 1410 FFS tape
ffffff  = tape serial number for the 1410 data base.
```

Section 5

FILE LOAD/UNLOAD UTILITIES

The following paragraphs provide the job control language and necessary information required to transfer files to and from the Sequential Access and Indexed Sequential Access Methods (SAM and ISAM) and to and from the Sequential Access and Virtual Storage Access Methods (SAM and VSAM). These utilities are effectively automatic and require only the JCL stream for control.

5.1 SAM to ISAM or VSAM Utility - UTBLDISM

This utility builds an MIPS ISAM data base or a MIPS VSAM data base from a MIPS SAM data base. Under control of the CC symbolic parameter on the EXEC card it can optionally build the ISAM or VSAM data base in the compressed and/or compacted form or reverse the process to produce the ISAM or VSAM data base in the standard form (see section 5.3).

The JCL DD cards used are:

//DATAFILE	DD	Parameters defining the new ISAM data base
//SAMFILE	DD	Parameters defining the existing SAM data base
//VSMFILE	DD	Parameters defining the new VSAM data base.

The new ISAM MIPS data base will be created with the following attributes: master index, cylinder overflow, independent overflow, delete option, write check and feedback reorganization criteria.

The procedure XSTOIS is used to build a NIPS 360 PFS ISAM or VSAM data base from a NIPS 360 PFS SAM data base. It is assumed that the existing NIPS SAM data base resides on a 9-track tape, and that the ISAM data base will reside on a 2314 disk pack with a disposition of KEEP. If output is to be a VSAM data base, it must have been previously defined via the VSAM service routine IDCAMS and cataloged on a VSAM user catalog. Using symbolic parameters for this procedure, one would use the following JCL:

```
//JOBNAME      JOB    (standard parameters)
//            EXEC   XSTOIS,ISAM=aaaaaaa,VISAM='SER=bbbbbb', X
//     PRIME=cc,INDEX=d,SAM=eeeeeee,VSAM='SER=ffffff',
//      CC=gggggg
/*
```

where:

aaaaaaa = name used for the new ISAM data base

bbbbbbb = disk serial number for the new ISAM data base

cc = number of cylinders of prime space for the new ISAM data base

d = size of index needed for the new ISAM data base

eeeeeeee = name of the existing SAM data base

ffffff = serial number of the tape that contains the existing SAM data base.

gggggg = parameter option for data record transformation (COMPRESS, COMPACT, EXPAND or both COMPRESS and COMPACT) and pad record suppression (NOPAD). Multiple values must be separated by commas and the whole parameter enclosed in single quotes. (see section 5.3).

For SAM to VSAM, replace the ISAM parameters with:

```
VSCAT=aaaaaaaa,VSDSN=bbbbbbbb,NONVSM='DUMMY,'
```

where:

aaaaaaaa = name of the VSAM user catalog

bbbbbbbb = name of the VSAM data base to be loaded.

Normally, after the file has been copied to disk, the remaining prime space will be filled with pad records. These are ISAM records with valid keys which have the delete byte set on. These records make it possible to add records to the end of the file during later update and still remain in the prime area.

If the user does not want these pad records, he may include the NOPAD keyword in the symbolic parameter CC on the EXEC card.

An example might be a SAM file that is dumped to disk for use with TP. In this case, it would be desirable not to pass all of the pad records for a query, while the user is waiting at the terminal.

Upon completion of UTBLDISM, the user will receive either the message,

'NIPS ISAM (VSAM) DATA FILE HAS BEEN SUCCESSFULLY CREATED'

if the job was successful, or a USER 0200 ABEND if the job was unsuccessful.

5.2 ISAM or VSAM to SAM Utility - UTBLDSAM

This utility builds a SAM data base from either an ISAM, a VSAM or another SAM data base. Under control of the CC symbolic parameter on the EXEC card, it can optionally build the output SAM data base in the compressed and/or compacted form.

The JCL DD cards used are:

```
//DATAFILE DD   parameters defining existing ISAM data file

//VSMFILE  DD  parameters define existing VSAM data file

//SAMFILE  DD   parameters defining existing SAM data file

//SAMOUT   DD   parameters defining new SAM data file
```

The procedure XISTOS is used to build a NIPS 360 PFS SAM data file from an ISAM or VSAM data file or copy a NIPS 360 SAM data file. It is assumed that the existing ISAM data file resides on a 2314 disk pack or that the existing SAM data file resides on a 9-track tape with standard label. If input is a VSAM data file, it must be cataloged on a VSAM user catalog. It is also assumed that the output SAM data file will be written on a standard label 9-track tape.

Using symbolic parameters for this procedure, one would use the following JCL:

```
//JOBNAME     JOB(Standard parameters)
//           EXEC XISTOS,ISAM=aaaaaa,VISAM='SER=bbbbbb',
//           SAM=eeeeee,VSAM='SER=ffffff',CC=gggggg
/*
```

where:

aaaaaa = name of the existing ISAM data file

bbbbbb = serial number of the disk volume where the ISAM data file resides

eeeeee = name used for the new SAM data file

ffffff = serial number for the new SAM data file

gggggg = parameter option for data record transformation (COMPRESS, COMPACT, EXPAND or both COMPRESS and COMPACT separated by a comma) and pad record suppression (NOPAD). Multiple values must be separated by commas and the whole parameter enclosed in single quotes.

For VSAM to SAM, replace the ISAM parameters with:

VSCAT=aaaaaaaa,VSDSN=bbbbbbbb

where:

aaaaaaaa = name of VSAM user catalog

bbbbbbbb = name of exisitng VSAM data file.

For SAM to SAM one would use the following JCL:

```
//JOBNAME   JOB       (Standard parameters)
//          EXEC XISTOS,OLDSAM=aaaaaa,OLDVSAM='SER=bbbbbb',
//          SAM=eeeeee,VSAM='SER=ffffff',PARM=gggggg
```

where:

aaaaaa = name of the existing SAM data file

bbbbbb = serial number of the volume where the SAM data file resides

eeeeee = name used for the new SAM data file

ffffff = serial number for the new SAM data file

gggggg = parameter option for data record transformation (COMPRESS, COMPACT, EXPAND or both COMPRESS and COMPACT separated by a comma). Multiple values must be separated by commas and the whole parameter enclosed in single quotes. (see section 5.3).

Keywords in the PARM are used to designate the form of the output SAM data base irrespective of the form of the input ISAM, SAM or VSAM data base. The keywords for the PARM and their implications are the same as those for the SAM to ISAM utility.

Upon successful completion of UTBLDSAM, the user will receive the message:

'NIPS SAM DATA FILE HAS BEEN SUCCESSFULLY CREATED'.

Prior to termination of the utility, it will print statistics that are beneficial for the user/analyst.

The first line of statistics printed gives the file name, volume serial number, date, time, and page number. Then information is printed concerning the sets.

There are seven columns of information displayed, with headings, as follows:

a.	SET	The first entry is "FIXED SET" and following is each periodic set with the number specified in the FPT.
b.	MINIMUM SUBSET SIZE (BYTE)	Shows the size, in bytes, of the smallest subset within the specified set, fixed or periodic. If a variable field is specified, the minimum and maximum size will not be the same.
c.	MAXIMUM SUBSET SIZE (BYTES)	Shows the size, in bytes, of the largest subset within the specified set, fixed or periodic. If a variable field is specified, the minimum and maximum size will not be the same.
d.	NUMBER OF SUBSETS PER DATA RECORD MINIMUM MAXMIMUM	These two columns reflect the minimum and maximum number of subsets within a periodic for the total record. The fixed set would be 1 for minimum and maximum. The absolute minimum for periodics would be Ø and the maximum, any variable number.
e.	TOTAL NUMBER OF SUBSETS	The total number of subsets in each periodic set for the entire file is printed here. The number printed for the fixed set is the total number of data records in the file.
f.	SIZE OF LARGEST SET	This field shows the maximum size in bytes of the indicated set for any record in the file.

For the ISAM file, the above-mentioned statistics are printed along with information on the organization of the ISAM file.

The number of PRIME, OVERFLOW, DELETE, and PAD records is calculated and printed. Also information from the DSCB is printed. The DSCB information concerns the INDEX, PRIME, and OVERFLOW cylinder/track allocation and usage. The column headings are "CYL/TRACKS ALLOCATED," "TRACKS UTILIZED," and "PERCENT TRACKS UTILIZED." The number of tracks in each cylinder overflow area is also provided. With this information the user can calculate the amount of space needed for his job. This alleviates unnecessary pad records that are added if any prime area remains.

5.3 Compression and Compaction of Data Records

Compression and compaction provide a means for the reduction of intermediate storage requirements for data without altering the integrity of the data. This data reduction scheme is particularly suited to data files that contain strings of identical characters or a large quantity of alphabetic data.

A string of identical characters is compressed by translating it to two bytes. The first byte is a control byte which indicates that compression has been applied and gives a count of the number of identical consecutive bytes that were in the original string. The second byte is identical to those in the original string.

A string of alphabetic characters is compacted by translating it to a control byte followed by a string of coded characters. The control byte indicates that compaction has been applied and gives a count of the coded characters. Each coded character represents a combination of two adjacent alphabetic characters.

Other NIPS components recognize compressed/compacted data and expand it prior to processing. File Maintenance (FM) will recompress and/or recompact an updated record before writing the record to the data file. Compression/compaction offers reduction in I/O time and

space requirements for the data file at the expanse or processing overhead.

Keywords in the PARM parameter (specified by the CC symbolic parameter) are used to designate the form of the output ISAM or VSAM data base irrespective of the form of the input SAM data base. The keywords for the PARM and their implications are as follows:

COMPRESS	-	compression is applied to the output data records.
COMPACT	-	compaction is applied to the output data records.
COMPRESS, COMPACT	-	a combination of compression and compaction is applied to the output data records.
EXPAND	-	compression and/or compaction is reversed to produce standard form data records.

If none of the parameters is specified, no data transformation takes place, i.e. the output ISAM or VSAM data base form will be the same as the input SAM data base form.

Section 6

UTQRTQDF

The utility program UTQRTQDF allows the user/analyst to create a NIPS 360 FPS data file from the answer file (QRT/QDF) produced by the retrieval processor (RASP). The output from this utility run is a sequential access method (SAM) file which is identical to a file generated using the file maintenance processor. The copy process is complete, carrying forward all of the logic statements associated with the original file as well as the FFT.

6.1 Input

Input for the cataloged procedure XQRTQDF comes from two sources. The first source is the paired data sets, QRT and QDF which are the normal output of RASP. The job setup description provides the proper assignment of these data sets to the utility. The second source is a single control card. This control card is free format, in that the parameter string may be initiated in any card column, and the parameters are separated by blanks or commas. Only one control card is permitted for each execution of the utility. The parameters provided on the control card are either one or two keywords and their corresponding operands. The first keyword is QUERYNO=. This keyword may be entered in any of the following forms:

QUERYNO=

QUERY=

Q=

and must be immediately followed (no intervening blank) by the retrieval number (answer identification number)

specified in the retrieval run creating the answer data sets. The query number is limited to four digits in size. Leading zeros may or may not be included in the number.

The second keyword parameter is RITID=, and may be entered in any of the following forms:

RITID=

RIT=

R=

and must be immediately followed by the report instruction table identification (RITID). This keyword and its operand may be omitted if the specific answer set desired did not use the RITID parameter for identification in the original RASP run.

The following examples indicate the types of control statements expected by the processor. First, an example of the input required to identify retrieval number (query number) 3 of a batched retrieval run, but with no RITID specified in the retriever input stream:

```
QUERYNO=3
   or
QUERYNO=0003
   or
QUERYNO=003  RITID=
```

Second, an example of the use of the RITID statement:

```
QUERYNO=3 RITID=UNAMEIT
```

6.2 Restrictions

The user may not attempt to output a data file from an answer stream formed under the following conditions:

a. RASP execution was a multifile run.

b. A sort was specified for the answer set identified.

c. Multiple IF condition statements were included in the retrieval number/RITID identified answer set.

d. Multiple SELECT statements were included in the retrieval number/RITID identified answer set.

Any batched retrieval condition that would cause resequencing of the resultant answer set, or any retrieval condition that would cause duplicate output records identified by a common retrieval number/RITID may not be used to create a file through the use of this utility.

6.3 Job Setup

The following statements illustrate the job setup used to execute the XQRTQDF procedure:

```
//          EXEC XRASP, parameters for user's file
//RASP.SYSIN DD *
  (RASP statement conforming to rules outlined
   under 6.1 for answer set to be processed by
   UTQRTQDF)
/*
//          EXEC XQRTQDF,SAM=MYFILE
//QRTQDF.SYSIN  DD  *
      (Control statement defining input as
       described in 6.1 and 7.1.1)
/*
```

The user obtains his new data file (MYFILE) on a standard-labeled, 9-track tape written according to the installation assigned density. The user may override the symbolic parameters to obtain input and output data sets differing from the procedure-specified standards.

The XQRTQDF procedure may be executed in a separate job from the RASP if the QRT and QDF are saved. In this case,

the QRTFILE and QDFILE symbolic parameters would have to be specified with the appropriate values.

6.4 Error Conditions and Processing

All error conditions sensed by this program will cause termination with a "user" abnormal end code of 1, 2, or 3. These codes are always accompanied by an error message uniquely identifying the error condition encountered. The following material is included as a quick reference of error listings. For full meaning of the error messages, see Volume IX, Error Codes.

INVALID KEY WORD - RUN TERMINATED

NO ANSWER SET FOR QUERYNO/RITID

QRT/QDF IS MULTIFILE PRODUCT - ERROR

NO FILE RECORD ON QRT - BAD INPUT

ONLY ONE CONTROL CARD PERMITTED

BAD RECORD TYPE FOUND - RUN TERMINATED

EOD FOUND COPYING FFT AND LS - NO DATA

USER-SPECIFIED SORT - CAN'T PROCESS

MULTIPLE IF/SELECT STATEMENTS USED WITH QUERY/RITID SPECIFIED - RUN ABORTED

ERROR READING QRT - ERROR ABEND

PHYSICAL EODAD ON QDF - ERROR ABEND.

NO RECORD FOUND - RUN TERMINATED

RECID SPECIFIED IN QRT CANNOT BE FOUND IN QDF

WARNING - THIS FILE BUILT FROM QRT/QDF WITHOUT AN FFT

Section 7

UTSUBCHK

The UTSUBCHK utility can be used to check out user-written subroutines (ALC, FORTRAN, COBOL) written to supplement NIPS applications. SUBCHECK links to the user-designated subroutines stored as a member on a partitioned data set in load form, prints user run specifications, prints subroutine input and output, and relates the success of the subroutine for each user input.

UTSUBCHK simulates the call initiated by the various NIPS application modules. The conditions, described in Volume I, Introduction to File Concepts, Development of Conversion Subroutines, are duplicated.

7.1 Input

Input to SUBCHECK is on punched cards. The first card in relates the run control parameters as follows:

Card Columns	Description
1-8 (REQUIRED)	User subroutine name, left-justified with trailing blanks as required.
10 (OPTIONAL)	Blanks - Alphameric input; do not convert. B - Convert data to binary prior to subroutine link.
12-14 (OPTIONAL) (fixed length argument)	A 3-digit number (use leading zeros if necessary). If column 10 is blank, number=length of data, starting in column 1 of data cards, to get and pass.

If column 10 contains a "B", number = length of data, starting in column 1 of data cards, to get and convert.

7.2 Functioning and Restrictions

If the card contains only the subroutine name, UTSUBCHK will assume variable length data input (not to exceed 80 bytes in length) with no conversion to binary required. In this case the argument length is determined by a scan, right to left, of each data card for the first nonblank character. If binary conversion is required, the resultant argument will be one full word in all cases.

All user data cards follow the control card. The data used for the argument must start in column 1. To allow for trailing blanks, the fixed length entry in control card columns 12 through 14 can be used.

The parameter list built starts with a half-word containing the binary length of the argument. A 256-byte intermediate work area is set up for the argument and function, contiguous to this length entry. This means that while either the argument or the function length may be up to 255 bytes, the sum of their lengths must be less than or equal to 256 bytes.

The entire input (unconverted) argument is printed out for each data card. Immediately following each of these entries, the subroutine output and results follow respectively.

Because the input can be up to 255 bytes several input cards might be required for each input argument. For fixed length arguments the same number of input cards are required for each argument (i.e., for a 173-byte argument three data cards are required for each argument - use blank cards as filler if necessary).

7.3 Job Setup

The following statements illustrate the job setup used to execute the XSUBCHK procedure:

```
//       EXEC   XSUBCHK,LIB=TESTER
//SUBCHK.SYSIN  DD *

(Control card defining input as described in 7.1)

data card(s)
/*
```

The LIB symbolic parameter defines the partitioned data set where the subroutine's executable load module is stored. The library need not be a WIPS file library, but it should have the same DCB attributes.

Sample Control Card

```
MYSUB              099
```

Sample data cards in the above case

```
                                    Col 80
1234567890AQFT86789ABCB ..... 123796

7890567893BCBDEPGHQ

1379654321BDLC86789DPQC ..... 569324

8432976843APQDETGHQ
```

Section 8

UTDMPLIB

The utility program UTDMPLIB allows the user/analyst to print the names of reports and/or logic statements associated with a NIPS 360 FPS data file. The output from this utility is a formatted listing. The input data file may be in Sequential Access Method (SAM), Indexed Sequential Access Method (ISAM) or Virtual Storage Access Method (VSAM) form.

8.1 Input

Input for the cataloged procedure XDMPLIB consists of the user's file (SAM, ISAM or VSAM) and a single control card. The control card is in a fixed format in that it must begin in card column 1 and parameters must be separated by commas. The control card is to be prepared as follows:

```
PRINT,REPORT,XXXXXX
```

where XXXXXX will contain one of the following parameters:

ALL	- provides a listing of all report names and all logic statements.
LIST	- provides a list of all report names.
(Report ID)	- provides a listing of all logic statement names for the report ID specified.

8.2 Job Setup

The following statements illustrate the job setup required to execute the XDMPLIB procedure:

```
//       EXEC   XDMPLIB,ISAM=TESTER
//UTDMP.SYSIN   DD  *
PRINT,REPORT,ALL
/*
```

The preceding example lists all report and statement names on a cataloged ISAM data set named TESTER.

```
//  EXEC  XDMPLIB,VSCAT='NIPS.CAT',ISAM='VSAM.TESTER'
//UTDMP.DATAFILE  DD  AMP='AMORG'
//UTDMP.SYSIN  DD  *
PRINT,REPORT,ALL
/*
```

The preceding example lists all report and statement names on the VSAM file VSAM.TESTER cataloged on NIPS.CAT.

```
//       EXEC  XDMPLIB,ISAM=TESTER,VISAM='SER=FFSLIB'
//UTDMP.SYSIN     DD  *
PRINT,REPORT,REPTA
/*
```

This example lists associated logic statement names for a report entitled REPTA included in the uncataloged ISAM data set TESTER residing on a 2314 volume labeled FFSLIB.

```
//       EXEC  XDMPLIB,SAM=TESTER,VSAM='SER=N12345'
//UTDMP.SYSIN     DD  *
PRINT,REPORT,LIST
/*
```

The preceding example provides a listing of all report names contained within the data set TESTER which resides on a 9-track, 800 bpi tape with the volume serial number of N12345.

The XDMPLIB procedure may be executed in a separate job or as a step of, for example, an FM update. It is to be noted that all routines used in conjunction with the execution of this procedure reside on FFS.JOBLIB.

Section 9

UTCLASS

The program UTCLASS provides the user with the capability of changing the classification of a NIPS 360 PPS ISAM, SAM or VSAM file. A single card is required which will contain only the new classification of the file.

9.1 Input

The program UTCLASS requires a single input card containing the new classification of the data file. The card is free format and the new classification will be left-justified with trailing blanks, if required, when written out to the file. The length of the classification field is 32 characters and truncation will be performed to the right.

To change classification of the file to blanks, that is, no classification, the input card must contain at least one blank enclosed in apostrophes. This is the only case in which an apostrophe which is entered on the input card does not become part of the actual file classification.

9.2 Output

Two outputs are produced by a successful run of this program:

a. The updated data file containing the new classification.

b. A listing on the printer indicating successful or unsuccessful updating and the new classification if successful.

9.3 Job Setup

The following statements illustrate the deck setup used to execute the UTCLASS procedure.

```
//        EXEC   XCLASS,ISAM=TESTER,
//        VISAM='SER=FPSLIB'
//CLASS.SYSIN DD  *

     (User-supplied card containing new
     classification)

/*
```

The preceding examples will change the classification on an uncataloged ISAM file.

```
//     EXEC XCLASS,SAM=TESTER,VSMOUT='SER=TSTVOL',
//          SAMOUT=

     (User-supplied card containing new
     classification)

/*
```

The preceding example will change the classification of a cataloged SAM file on tape.

```
//     EXEC   XCLASS,SAM=TESTER,
//        USAM=2314,VSAM='SER=FPSLIB'
//CLASS.SYSIN DD *

     (User-supplied card containing new
     classification)

/*
```

The preceding example will change the classification on an uncataloged SAM file on disk.

```
//  EXEC  XCLASS,VSCAT='NIPS.CAT',ISAM='VSAM.TESTER'
//CLASS.DATAFILE  DD  AMP='AMORG'
//CLASS.SYSIN  DD  *
     (user-supplied card containing new classification)
/*
```

The preceding example will change the classification of the VSAM file, VSAM.TESTER, cataloged on NIPS.CAT.

Section 10

SOURCE LANGUAGE STORAGE

Source programs for NIPS components may be stored on a library to facilitate housekeeping and program maintenance. Maintaining a library will ensure that a single, current version of the source program is always available. The Source Program Library can be updated by batch jobs, or by the EDIT component if on-line terminals and the NIPS TP component are available. EDIT capabilities are described in Volume VI of the NIPS User Manuals, Terminal Processing (TP).

10.1 Source Libraries

Two types of libraries can be utilized for storage. Both types are direct-access partitioned data sets. The first type of library is normally used to store 80-character card images. This library is similar to a Procedure or Macro Library and has fixed length, 80-character records, normally blocked to an efficient blocksize which is a multiple of 80.

The second type of library is the File Library. This library has records of undefined length and is normally used to store the compiled, executable NIPS user programs (Retrievals, RITs, etc.). Source programs will be stored on this library in 800-character blocks, containing 10 card images each. OS utilities may not be used to modify source on this type of library, but NIPS components may as discussed below.

Each source program member stored on a library by a NIPS language component or utility will contain an indicator as to which NIPS language component (RASP, PM, etc.) is involved. This will enable a terminal user to scan a library for RITs, logic statements, etc.

10.2 Means of Storing Source

From the batch, source programs may be stored on a library by either of two means. The NIPS component which compiles the source program can be requested to place the source on a specified library, or the program, UTSOURC, may be executed as a stand-alone utility.

10.3 Conditions of Source Library Update

Source will be stored on a library by the UTSOURC utility and the NIPS language components if the following two conditions are met:

a. A DD statement specifying the user's library and having a DDNAME of SOURCLIB is present in the job step JCL.

b. A source control card is placed immediately in front of the source program in the input stream.

The SOURCLIB DD statement is included in each NIPS procedure which allows compilation or structuring of source material. It is also included in the XUTSOURC procedure which executes the UTSOURC utility. The user's library name is specified through use of symbolic parameters.

10.4 Source Control Statement

The source control card follows the general format of IEBUPDTE utility control cards. Any number of blanks may separate the fields. The format is as follows:

```
./ [ADD   ]  [NAME=xxxx]  [FS   OP  ]
   [REPL  ]               [FB   QUIP]
   [DELETE]               [RASP      ]
```

The ./ must be coded in columns 1 and 2. These two characters identify this card image as a source control statement.

Either ADD, REPL, or DELETE must be coded if NAME is coded. ADD says that this is a new source member that is to be added to the library. If a member already exists with the same name, a diagnostic message will be printed, and no library action will take place. REPL will replace a member on the library with the new source which follows the control card. If no member presently exists with the same name, it will be added. DELETE will delete the member with the name specified.

NAME gives the name of the source program assigned by the user. The name must follow standard OS 360 rules for library member names and may be up to eight characters in length.

The last operand on the control statement is used to specify the NIPS component to which the source number is related. That is, a source RIT will have OP coded in its source control statement. This field is optional when a NIPS language component is used to update a source library since the appropriate component ID will automatically be included. It must be included if the stand-alone utility is used and NIPS source material is being stored. If other data is being stored, the 4-byte value entered is copied so that classes of data can be accessed by EDIT.

The existence of a source control card containing no operands (blank except for ./ in columns 1 and 2), will signal the end of a source member and can be used when additional source material exists that will not be added to the library.

10.5 Source Member Names

The name given to the source on the source control card does not have to be the same as that assigned on the TITLE, CREATE, or ADD statement card within the source. It is suggested that the user adopt a naming convention such as adding an S suffix to the name of the NIPS program. This will provide a ready recognition of the source member.

10.6 Operation of Source Library Update

The input stream to a NIPS language component or the utility may consist of one or more source programs to be stored on a library. The library action requested by the source control cards is performed as the input stream is read initially. No library action is taken on data in the input stream until a source control card is encountered. If the control card specifies a delete action, it is performed immediately and reading of the input resumes. If an ADD statement is encountered, the library is checked for existence of a member with the same name. If one is found, a diagnostic is printed, and no library action is taken with that member. If a duplicate member name is not found or a REPL action is specified, the following cards of the input stream, up to the next source control card or end of input, are placed on the library and given the member name specified.

When using a NIPS language component to update a source library, the source control cards will not be passed to the language translator phases and will thus not interfere with normal compilation or action of NIPS components.

10.7 Sequencing of Source Material

All source material that is added to a library will be sequenced in columns 73-80 and printed during the library action. If a NIPS language component is being used, the newly sequenced source will be passed to the language translator phase.

10.8 Listing Source Library Members

The UTSOURC utility may also be used to list a source member from a source library. This is done by specifying the library using symbolic parameters and the names of members to be listed using the PARM value.

10.9 Job Setup

If a MIPS language component is used, the procedure contains the SOURCLIB DD statement. Specify the name of your library through use of the symbolic parameters as outlined in Volume VIII, Job Preparation.

The input stream may be a member in your library. If so, use the symbolic parameter NAME to specify the member name. Otherwise override the XUTSOURC SYSIN DD statement to define the input stream. Note that SYSIN is ignored for list functions.

If the stand-alone utility is used, the following statements illustrate the deck setup used to update a source library:

```
//JOBNAME      JOB          (Installation Parameters)
//JOBLIB   DD               (Installation JOBLIB Parameters)
//UPD    EXEC   XUTSOURC,SOURCL=YOURLIB
//SOURC.SYSIN   DD   *
   Source control statements and source members
/*
```

The following statements illustrate the deck setup used to list a member from a source library:

```
//JOBNAME   JOB             (Installation Parameters)
//JOBLIB     DD             (Installation JOBLIB Parameters)
//LIST  EXEC   XUTSOURC,SOURCL=YOURLIB,PARM.SOURC=SRCNAM
/*
```

Section 11

INDEX SPECIFIER (UTNDXSPC)

The Index Specification utility allows a user to specify indexing information for a data file without running a File Structure or File Maintenance job. The user can add and delete indexes in the same run. For each index added, all information necessary to make that index operational is generated and placed in the Index Data Set.

Any fixed-length field defined in the file may be specified as a secondary index. Any variable field, variable set or fixed-length alpha field defined in the file may be specified as a keyword index. A fixed-length field may not be specified as both a secondary and keyword index.

UTNDXSPC acts as the driver to perform the functions of calling Index Specification to insert or delete Index Descriptor Records in the data file, and of executing Index Maintenance to correlate the Index Descriptor Records in the data file with the Index Control Records in the Index Data Set and to update the latter accordingly. UTNDXSPC operates on an ISAM, SAM or VSAM data file.

11.1 UTNDXSPC Input

UTNDXSPC accepts SUB/TAB and INDEX statements as input. These statements must be submitted through the SYSIN device. Further discussion of Index Specification may be found in Volume II, File Structuring.

11.1.1 SUB/TAB Card

A SUB/TAB card is used to describe a subroutine or table to be used by secondary indexing. A subroutine or table may be a conversion routine, to convert data from an internal data file format to a separate index format. On the other

hand, a subroutine or table may be an analyzer routine, designed to analyze the parameter list of a FUNCTION operator to determine index usage and provide a list of values for index qualification.

For a keyword indexed field, a subroutine or table defines options that direct the selection of keyword values. It may designate a stop word table which is a list of irrelevant (noise) words, a dictionary which is a list of all non-literal keywords, or a user scan routine, which is a subroutine provided by the user to process the keyword indexed fields.

One SUB/TAB card must be submitted for each subroutine referred to by the INDEX statements. A subroutine or table may be defined for one function only in an Index Specification run, and each unique subroutine or table name may be submitted only once per run.

All SUB/TAB statements must appear first in the input stream, before the INDEX statements.

The SUB/TAB statement is free-format. The operands must be in order, each separated from the others by at least one blank. A period is required after the last entry to signify the end of the statement.

```
{SUB       }          {STOP }
{SUBROUTINE} subname  {DICT }
{TAB       }          {SCAN }
{TABLE     }
                      {ANALYZE}          [ALPHA  ]
                      {CONVERT} nnn-nn   [BINARY ]
                                         [COORD  ]
                                         [DECIMAL]
```

a. Statement Identifier

SUB or SUBROUTINE - subroutine statement

TAB or TABLE - table statement

b. subname - subroutine or table name

c. Function Identifier

For Keyword Indexes:

STOP - subname to be used with Stop Word Table

DICT (or DICTIONARY) - subname to be used with Dictionary

SCAN - subname is a user scan routine
This is the last operand for a Keyword SUB/TAB statement.

For Secondary Indexes:

ANALYZE - specifies conversion of data to index format

CONVERT - specifies analysis of index usage

The following operands are required for the CONVERT function.

d. nnn One-, 2-, or 3-digit field specifying the length in bytes of the input to the table or subroutine.

e. nnn One- or 2-digit field specifying the length in bytes of the output produced by the table or subroutine (maximum is 30).

f. ALPHA
BINARY
COORD
DECIMAL
Mode of data output by the subroutine or table

Example of a CONVERT statement:

```
SUB  convsub  CONVERT  17  2  ALPHA.
```

Example of an ANALYZE statement:

```
SUB  analsub   ANALYZE.
```

Example of a DICTIONARY statement:

```
TAB  DICTNAME  DICT.
```

11.1.2 INDEX Statement

An INDEX statement must be provided for each data file field to be designated as an index and for each index field to be deleted. The delete operation does not remove the field from the file; it merely eliminates the option of indexing the file on the contents of that field.

Like the SUB/TAB statement, the INDEX statement is free-format. Unless otherwise indicated the operands are order-dependent. Each must be separated from the others by at least one blank. A period is required after the last operand to signify the end of the statement.

```
INDEX fieldname [ADD    ]
                [DELETE ]     convsubname analysubname

KEYWORD stopname dictname scanname [TEXT    ]
                                   [SEPARATE]
                                   [DROP    ]
                                   [RETAIN  ]
```

a. Statement Identifier - INDEX

b. fieldname — FFT name of field or group to be indexed

c. Action Indicator

ADD - field is to be added as an index. This is the default value. However, it is required. If a conversion and/or

analyzer subroutine/table, or KEYWORD indexing is defined.

DELETE- field is to be removed as an index. the data field itself is not affected. This is the last operand required for delete action.

The following two optional operands are used only with secondary indexes.

d.	convsubname	Name of the table or subroutine to convert data data from datafile format to index data set format.
e.	analysubname	Name of subroutine to be used to analyze a FUNCTION operator parameter list and determine index usage.

NOTE: If both conversion and analyzer subroutines are specified, the conversion subroutine must be specified first. If only one is specified, its function will be determined from the parameters on the SUB/TAB statement. An analyzer subroutine or table may not be used as a conversion routine, and a conversion subroutine or table may not be used as an analyzer routine in the same Index Specification run.

The remaining operands are used only when defining keyword indexed fields:

f.	KEYWORD	Required to differentiate between the two types of indexed fields
g.	stopname	name of the Stopword Table (optional)

h.	dictname	name of the Dictionary (optional)
i.	scanname	name of user scan routine (optional)
j.	Hyphen Option	
	TEXT	No text editing. Hyphens remain as they appear in the text. The default is TEXT.
	SEPARATE	Hyphens are always treated as word separators and are replaced with blanks. Hypenated words are treated as two or more separate words.
	DROP	Hyphens are always dropped. If a hyphen is preceded by a text character and followed by a text character or one or more blanks and a text character, the hyphenated text is connected to the following text string without the hyphen.
	RETAIN	The opposite of DROP. Hyphens are always retained. As with DROP, hyphenated text is connected with following text, but thy hyphen is retained as an embedded character of the whole word.

Any combination of stopname, dictname, scanname and hyphen option may be specified in any order.

Example of an Add Index Action:

```
INDEX  MEQPT  ADD  CONSUB.
```

Example of a Delete Index Action:

```
INDEX CNTRY DELETE.
```

Example of an Add Keyword Index Action:

```
INDEX ANAME ADD KEYWORD RETAIN
     SCANNER STOPPER KEYWDS.
```

11.2 UTNDXSPC Output

UTNDXSPC builds and inserts Index Descriptor Records into the data file for indexes added and deletes Index Descriptor Records for indexes deleted. It then calls Index Maintenance to either generate or update the Index Data Set. UTNDXSPC also lists a summary of actions performed plus any error conditions encountered.

11.3 UTNDXSPC Job Setup

The following JCL examples can be used to invoke the cataloged procedure XSP which will either generate or update a disk-resident Index Data Set based on the ISAM, SAM or VSAM data file.

The first example illustrates a situation in which an index data set is to be regenerated. The uncatalogued SAM data file, MYFILE, resides on a 9- track tape with standard labels and system default density. It contains Index Descriptor records in the FFT from previous runs. The new Index data set must have the same name as the data file (the system will suffix an X). The new index will have the default blocksize of 560 and default number of blocks of

200. Blocksize must be between 560 and 1020 bytes and at least 50 blocks must be allocated.

```
//SAMPLE1     JOB    (standard parameters)
//STEP1       EXEC XSP,SAM=MYFILE,VSAM='SER=MYTAPE',
//       INDEX=MYFILE,XVOL='SER=MYDISK',XDISP=NEW,
//       SAMOUT=,PARM=GEN
//SYSIN DD *
         (Index Specification statements)
/*
```

NOTE SAMOUT=, is required for SAM runs. PARM=GEN is required when generating a new Index Data Set and Index Descriptor records exist in the data file FFT.

In the following example, index specifications on the catalogued index data set MYFILEX and the ISAM data file MYFILE will be updated:

```
//SAMPLE2     JOB    (standard parameters)
//       EXEC   XSP,ISAM=MYFILE,XINDEX=MYFILE
//SYSIN DD *
         (Index Specification statements)
/*
```

The following will update the index specification for the VSAM file, VSAM.MYFILE, catalogued on NIPS.CAT:

```
//SAMPLE3     JOB    (standard parameters)
//       EXEC XSP,ISAM='VSAM.MYFILE',XINDEX=MYFILE,
//       VSCAT='NIPS.CAT'
//UTXSP.NEWFILE   DD  AMP='AMORG'
//SYSIN DD *
         (Index Specification statements)
/*
```

Section 12

INDEX TRANSFER (UTNDXTFR)

An Index Data Set may reside on a direct-access device or on tape, but only the disk-resident medium can be used by any NIPS component. Index Transfer (UTNDXTFR) permits the user to transfer the entire data set, from one resident medium to the other.

The primary use of UTNDXTFR is to reorganize the disk-resident indexes. Initially, a disk-resident Index Data Set is packed with index information. As the indexes are maintained, gaps or unused areas may occur in the data set as records are deleted and others are added. By using UTNDXTFR, the user can transfer the disk-resident data (only the valid information is transferred) to tape, and again from tape back to disk. This operation condenses the data set. The tape so created may be retained as a backup.

In the disk to tape mode of operation, a statistical printout of unique values (for secondary indexed fields) and keywords (for keyword indexed fields) and their occurrences are optionally developed. Binary values will be converted to decimal for ease of reading. However, any dictionary fields using conversion subroutines or keyword fields having synonyms in the dictionary, will be printed in the converted form, just as they appear in the Index Data Set.

12.1 UTNDXTFR Input

Two cataloged procedures are available for invoking the UTNDXTFR utility. XTRDISK will transfer a disk-resident Index Data Set to a sequential access medium, while XTRTAPE will reconstruct an Index Data Set from a previously unloaded tape version of the disk data set.

12.2 UTNDXTFR Output

UTNDXTFR produces an Index Data Set on the residence medium indicated.

12.3 UTNDXTFR Job Setup

The following statements illustrate the deck setup used to unload the TESTERX Index data set from a 2314 disk pack to a 9-track unlabeled tape.

```
//JOBNAME     JOB   (standard parameters)
//STEPNAME    EXEC   XTRDISK,XFNAME=TESTERX,
//                   XFVOL='SER=MYPACK',XTNAME=INXSAM,
//                   XTVOL='SER=MYTAPE'
```

The following example illustrates the deck setup to be used for a VSAM file, VSAM.MYFILE, cataloged on NIPS.CAT:

```
//JOBNAME     JOB   (standard parameters)
//STEPNAME    EXEC  XTRDISK,XFNAME='VSAM.MYFILE',
//            XFVOL='SER=MYPACK',XTNAME='VSAM.INXSAM',
//            XTVOL='SER=MYTAPE',VSCAT='NIPS.CAT'
//XTR.DATAFILE DD AMP='AMORG'
```

The following example illustrates the deck setup to be used to reconstruct a disk resident index data set from a previously unloaded version of index data set.

```
//JOBNAME     JOB   (standard parameters)
//STEPNAME    EXEC   XTRTAPE,XTNAME=TESTERX,
//            XTVOL='SER=MYPACK',
//            XFNAME=INXSAM,XFVOL='SER=MYTAPE'
```

Section 13

UTFLDSCN

UTFLDSCN scans the NIPS components source statements and provides the user with the count of data fields referenced in the source statements. This utility is useful in helping the analyst to determine the activity of his data fields. This will assist the user in determining which fields are candidates for index fields in the Secondary Indexing capability.

The utility processes source statements pertaining to a single file in one execution. Multifile RITs and multifile queries will be accepted as input; however, only the data fields of the input data file will be processed. All other files will be ignored. In order to completely process multi-file RITs and multifile queries, it would be necessary to include the source statements in an execution for each file referenced.

UTFLDSCN outputs a listing of the source input statements followed by a listing of the count of references for the data fields and a summary listing of data field reference count for each batch component. A transaction record will also be output for each data field referenced in a single source input statement.

13.1 UTFLDSCN Input

Input to the utility consists of a NIPS data file in SAM, ISAM or VSAM format, a control card, and the source input statements and/or members of a partitioned data set.

The format of the input control card is as follows:

```
./ SOURCE  COMP=XXXX,NAME=SNAME,MEMBER=MNAME
```

where ./ must be in columns 1 and 2 followed by one or blanks.

COMP=XXXX where XXXX may be FM, RASP, OP, or QUIP to identify the component.

NAME=SNAME where SNAME is the name of the source statement. If the source is a logic statement, the name must be the report name and logic statement names enclosed in quotes. If the report name is less than seven characters, it must be padded with blanks to seven characters.

MEMBER=MNAME where MNAME is the member name of the source statement on a partitioned data set. If this operand is used, a partitioned data set must be included in the job stream. If this operand is omitted, the source statements must follow the control card in the input stream.

13.2 UTFLDSCN Output

The outputs from UTFLDSCN are as follows:

Source Listing - The source input records in input order.

Field Listing - This output consists of a header for each source deck indicating file name, component name, source name, and member name, if any. The body of the listing consists of only those fields referenced and the count of references. This will follow the source listing for each source module. After all source modules have been processed, a summary listing will be provided containing the count of the field references per component.

Transaction Data Set - The transaction data set will be 50 characters long with an 'S' in column one to be used as a logic statement name. The format is as follows:

```
Column  1    - CHARACTER 'S'
       2-8   - FILE NAME
       9-12  - COMPONENT NAME
      13-25  - SOURCE MODULE NAME
      26-33  - FIELD NAME
      34-36  - SET NUMBER, DECIMAL
      37-42  - COUNT OF REFERENCES, DECIMAL
      43-48  - DATE - MMDDYY
      49-50  - UNUSED
```

Parameters may be entered in the PARM field on the EXEC card to suppress output. They are as follows:

NS - Suppress printing of source input

NL - Suppress field and summary listing

NT - Suppress transaction output.

13.3 Job Setup

The utility is executed by a procedure, XUTFSCAN. If the input statements are members of a partitioned data set, the data set must be included in the job stream. The following statements illustrate the job setup used to execute the XUTFSCAN procedure against the ISAM data file TEST360:

```
//  EXEC XUTFSCAN,ISAM=TEST360,LIB=TEST360
//SYSIN  DD  *
./   SOURCE  COMP=FM,NAME='TESTO A',MEMBER=TESTA
./   SOURCE  COMP=RASP,NAME=TEST01
      RASP  Query Statements
        :
        :
./  SOURCE  COMP=OP,NAME=TESTFFF
     OP RIT Statements
       :
./  SOURCE  COMP=QUIP,NAME=QTEST,MEMBER=QTESTA
/*
```

The following JCL could be used to execute the XUTPSCAN procedure against the VSAM data file VSAM.TEST360:

```
// EXEC  XUTPSCAN,ISAM='VSAM.TEST360',LIB=TEST360,
//       VSCAT='HIPS.CAT'
//UTFLDSCN.DATAFILE  DD  AMP='AMORG'
//UTFLDSCN.SYSIN  DD  *
       (field scan control statements)
/*
```

Section 14

UTNDXKAN

UTNDXKAN provides the user with the capability to analyze the words in fields for which keyword indexing is to be specified. The results of the analysis can be used to determine the contents of stop word tables and dictionaries that are to be associated with those fields. If the system scan subroutine does not recover words as the user desires, a user-written scan subroutine can be used with this utility. All the words contained in a data base field can be displayed, or those words which are irrelevant (noise words) can be selected and compared to the words in the system stop word table. If that table is not adequate, the Dictionary Maintenance utility can be used to build a user stop word table. Then this utility is able to list all relevant (nonstop) words which can be used to determine dictionary requirements. If a dictionary is not employed, the words not in the stop word table will all appear as keyword entries in the index data set. Through dictionary application, synonomous words and words with varying suffixes can be collected under one Index Data Set entry, and the synonym or suffixed form can still be used in a query statement. The Dictionary Maintenance utility can be used to build a dictionary so that this utility can produce a list of the words which will become Index Data Set entries together with the keywords (including synonyms and suffixed words) associated with them which can be used as query arguments.

UTNDXKAN processes either a SAM, an ISAM or a VSAM data file. The fields that are to be processed are specified by control statements. For each field specified, the utility obtains from the control statement or from the file itself the names of the scan subroutine, stop word table and dictionary required to process that field. It scans all values in the data base (unless the number of records to be processed has been limited by the user), optionally matches the recovered words to a stop word table, then optionally matches the remaining words to a dictionary. The actual

functions performed are controlled by accepting names to FFT entry specifications or by specifying BYPASS. Override names or BYPASS must be specified for all functions if the FFT entry does not indicate keyword indexing.

UTNDXKAN displays word lists with record frequency counts or, optionally, record identifications for each field processed or for all fields as a group. Frequency count reflects the number of NIPS records in which a word appears at least once.

14.1 Input

UTNDXKAN accepts FILE and FIELD control statements. The FILE statement is optional. At least one field statement is required. Control statements are coded on cards or as card images and are contained in columns 1 through 71. Each statement must begin in a new record with the statement identifier FILE= or FIELD= in column 1 of that record. The file or field name shall immediately follow the statement identifier. A statement that exceeds 71 characters can be continued on one or more additional cards in columns 1 through 71. A nonblank character must be placed on column 72 to indicate continuation. A control statement can be interrupted after any comma or blank. Words may not be split between records. Column 72 of the last or only record of each control statement must be blank. Columns 73-80 of all records are ignored.

A control statement operand is made up of two or more keyword parameters. Each operand must be preceded and followed by one or more blanks or commas unless an operand terminates in column 71, in which case a continuation character (in column 72) may follow the operand. If multiple values are specified for an operand, at least one blank or comma must separate each value and the group of values must be enclosed in parentheses. The operands can be coded in any order. No extra commas are required to indicate omitted operands.

14.1.1 FILE Statement

The FILE statement may be omitted. It applies to file processing and affects all the fields to be processed. If used, it may appear only once and must be the first statement. Its operands allow control of file access and output merging.

The format of the FILE statement is as follows:

```
FILE=filename

     MERGE=YES                    BYPASS=[STOP    ]
                                         [NONKEY  ]
                                         [KEYWORD ]
                                         [SYNONYM ]
                                         [SUFFIXES]

     SKIP=nnnn              STOPAFT=nnnnn
```

a. Statement Identifier

   FILE=filename - must be coded in column 1; specifies the name of the file.

b. MERGE=YES

   Specifies that the word lists from all fields should be merged into one group of lists that reflects the entire file.

c. BYPASS=(option1,option2,...)

   Defines the word lists to be suppressed for all fields and overrides the FIELD statement BYPASS operand. Only valid if MERGE=YES is specified. The display list identified by the following terms are omitted from the output for all fields:

STOP	stop-word table matches
NONKEY	stop-word table and dictionary

non matches

KEYWORD	dictionary matches including synonym sublists
SYNONYM	keyword sublists
SUFFIXES	dictionary non matches which are composed of keywords with valid suffixes; the keywords appear in the keyword list.

d. SKIP=nnnnn

A number between 1- 32,767 of NIPS logical records to skip before processing any fields.

e. STOPAFT=nnnnn

A number between 1- 32,767 which specifies the maximum number of NIPS logical records to process before stopping.

14.1.2 FIELD Statement

The FIELD statement identifies a field to be analyzed. At least one FIELD statement is required and up to 50 are allowed.

The format of the FIELD statement is as follows:

FIELD=fieldname

SCAN=name

STOP= [BYPASS / name]

DICT= [BYPASS / name]

HYPEN= [TEXT / DROP / RETAIN / SEPARATE]

BYPASS=[STOP / NONKEY / KEYWORD / SYNONYM / SUFFIXES]

RECID=[NO / YES]

a. Statement Identifier

FIELD=fieldname - must be coded in column 1. Specifies the name of the field or variable set to be analyzed.

b. SCAN=name

Identifies the scan subroutine to be used instead of the FFT scan subroutine or the system scan subroutine.

c. STOP=[BYPASS / name]

Overrides the FFT stop-word table specification or specifies BYPASS to cause the stop-word table match function to be omitted for the field in which case all recovered words are nonstop words. The default is BYPASS.

d. DICT=[BYPASS / name]

Overrides the FFT dictionary specifications or specifies BYPASS to cause the dictionary match function to be omitted for this field in which case all nonstop words are keywords. The default is BYPASS.

e. HYPEN=[TEXT / DROP / RETAIN / SEPARATE]

Overrides the FPT option specification. Default value is TEXT. Discussion of HYPEN options may be found in section 11.1.2 of this volume.

f. BYPASS=(option1,option2...)

Not applicable if BYPASS was specified on a FILE statement. It defines the word lists to be suppressed for the field. The display lists identified by the following terms are omitted from the output for the field.

STOP	stop-word table matches
NONKEY	stop-word table and dictionary non matches
KEYWORD	dictionary matches including synonym sublists
SYNONYM	keyword sublists
SUFFIXES	dictionary non matches which are composed of keywords with valid suffixes; the keywords appear in the keyword list.

g. RECID= [NO / YES]

Specifies that record identification are to be shown in the word lists instead of frequency counts. The default is NO.

14.2 Output

UTNDXKAN displays one list of words with either frequency counts or major record identifications for each field processed or for all fields as one group if the merge option is specified. A 2-character code associated with each word identifies its type. Words from types for which

bypass is specified are omitted from the list. If a dictionary was specified and a data word matched a convert synonym or was suffixed, the dictionary word which will be substituted for the data word is inserted after the data word at an offset.

14.3 Job Setup

The following JCL statements illustrate the deck setup used to invoke the XKA cataloged procedure for a cataloged ISAM file and a cataloged user library:

```
//  EXEC XKA,ISAM=filename,LIB=libname
//XKA.SYSIN DD *
        (user-supplied control statements)
/*
```

where

filename - name of the NIPS ISAM data file

libname - name of library containing user scan subroutines, stop word tables and dictionaries.

The following JCL statements illustrate the deck setup used to invoke the XKA cataloged procedure for an uncataloged SAM file and an uncataloged user library:

```
//  EXEC XKA,SAM=filename,VSAM='SER=aaaaaa',
//     LIB=libname,VLIB='SER=bbbbbb'
//XKA.SYSIN DD *
        (user- supplied control statements)
/*
```

where

filename - name of the NIPS SAM data file

aaaaaa - serial number of the SAM data file

libname - name of the library containing user scan subroutines, stop word tables, and dictionaries

bbbbbb - serial number of the user library.

The following JCL statements illustrate the deck setup used to invoke the XKA cataloged procedure for a VSAM data file cataloged on a NIPS.CAT:

```
//  EXEC XKA,VSCAT='NIPS.CAT',
//   ISAM=filename,LIB=libname
//XKA.DATAFILE  DD  AMP='AMORG'
//XKA.SYSIN  DD  *
          (user-supplied control statements)
/*
```

Where:

filename - name of NIPS VSAM data file

libname - name of the library containing user scan subroutines, stop word tables, and dictionaries.

Section 15

DICTIONARY MAINTENANCE (UTNDXKMD)

UTNDXKMD creates, updates, and displays stop-word tables and dictionaries. During one execution it processes any number of tables provided all tables are stored in the same library. It accepts as input table statements which define functions to be performed and value statements which provide data for create and update functions. Each table statement defines one of the following functions.

- add stop words
- delete stop words (update)
- add keywords
- add suffixes
- add synonyms
- delete keywords (update)
- delete suffixes (update)
- delete synonyms (update)
- display stop word table
- display dictionary

In addition, a table statement may include parameters which define a table page size (create), dictionary display options, and value statement display and sequence check options. Since a table statement defines only one function, more than one table statement with its associated value statements will probably be required to create a dictionary or to update any table. For ease of reference, a table statement and its value statements will be called collectively a function statement.

Table Creation

Only one function statement, ADD=STOPWORDS, is required (or permitted) to create a stop word table. Value statement words should be those words which appear many times (high frequency) or which the user will never use as search terms. If the amount of data is small, a stop word table will probably not be required. If the amount of data is very large, the user would benefit by executing the Keyword Analysis utility to obtain word frequency counts. Words which appear in more than 25 percent of the data file records are good stop word table candidates.

Three function statements may be used to create a dictionary -- add keywords, add suffixes, and add synonyms. However, all three statements may be used to define dictionary words with suffix specifications. If no synonyms are to be defined, only the add keywords statement need be used; if only synonyms are to be defined, only the add synonym statement need be used. On the other hand, all three statements may be used to define the same word as a keyword (add keywords) whose ending changes when it is suffixed (add suffixes) and that is synonomous with another word (add synonyms). If more than one function statement is used, the statements may appear in any sequence in the input stream. In fact, function statements and display statements from any number of tables may be intermixed. The only sequence restriction occurs when the user chooses to have UTWDXKWD sequence check the value statements in a function statement. UTWDXKWD processes the input in phases: it processes each function statement as an independent unit, then it processes each unique table word from all function statements.

Dictionary words should normally be those words which change their form when suffixed or which are to be grouped under one term in the Index Data Set. However, low frequence words in a very large data base might also be included. The decision to exclude words by using a stop word table or to include them by using a dictionary is arbitrary. Using a stop word table should increase run time. However, when both tables are used, both must be changed to include a word which is a stop word.

Table Update

Add stop-word and delete stop-word statements are used to update a stop-word table. All other function statements are used to update a dictonary. The difference between create and update is that during update all input words are matched against existing tables. Note that an existing table may be created; in create mode, UTNDXKMD does not test for the existence of a table -- it creates a new table from input data that replaces the existing table.

UTNDXKMD does not include a delete-table function. Use the OS utility IEHPROGM to scratch each page (member).

Dictionary delete function statements are more specific than add statements. A delete keyword statement removes that word from a table; it also removes all changed-form suffix entries and synonym entries associated with the word. A delete suffixes statement removes only changed-form suffix entires; the root word and synonym entries remain unchanged. A delete synonym statement removes only synonym entries. Another difference between add and delete statements is that add synonym words must be grouped in parentheses to show synonomous relationships; delete synonym words are independent and require no parentheses.

15.1 UTNDXKMD Input

UTNDXKMD accepts table and value statements from the SYSIN device. Formats conform to general NIPS specifications. Statement entries are punched in free format in positions 1-71. Position 72 is used for table statement continuation (value statements are not continued). Positions 73-80 of value statements will be optionally sequence checked within each function. Blanks, commas, and parentheses are used as entry delimiters and may not be part of an entry; the equals symbol is used to identify table entries and may not be used for any other purpose. All other characters are assumed to be part of entry terms; literals are not recognized as such because they are irrelevant table entries.

Table Statements

Table statement entries are composed of keywords and operands separated by an equals symbol. With the exception of the display keyword, all operands are single terms. If two terms are used in the display keyword operand they must be enclosed in parentheses; otherwise no special notation is required with any operand.

The table type keyword (STOPWORD, DICTIONARY) identifies the beginning of a table statement. It must be punched beginning in column 1 of the input record. All other keywords and operands may be punched in any position in columns 1-71. It is permissible to punch a keyword in one record, an equals symbol in a second record, and the operand in a third record. No keyword or operand may be split between records, however. Any number of blanks and commas may appear between terms and around the equals symbol (and a parentheses if any are required). All records except the last (or only) record that contain table statement terms must include a nonblank punch in position 72 to indicate that the record is not the last (or only) one for the statement.

The table type keyword and the mode keyword (CREATE, UPDATE, DISPLAY) are required in all table statements. The operation keyword (ADD, DELETE) is required in all create and update mode table statements.

All other keywords are optional. In create mode, the user can specify a maximum table page size (PAGE). If he does not, a 1K default size will be used. Note that if input data for a table does not completely fill one page, the actual page size will be equal to the space used, not the maximum size. However, the maximum size will be carried in the table directory and will be considered each time the table is updated. In create and update mode statements, the user can request that all value statements be sequence checked (SEQCK) and/or displayed for diagnostic reference (DIAGNOSTIC). In display mode for a dictionary, the user can select various word lists (DISPLAY).

A table cannot be created and updated in the same run. It is impossible to display a table before it is updated in the same run (statements are sorted before they are processed). Only one display mode statement may be present for a stop word table; any number may be present for a dictionary provided the DISPLAY keyword operands for each are unique.

The format of table statement entries is as follows:

```
{STOPWORD=   }  table name        MODE=[CREATE ]
{DICTIONARY= }                         [UPDATE ]
                                       [DISPLAY]

{ADD=   }  [STOPWORDS]                 PAGE=nnk
{DELETE=}  [KEYWORDS ]
           [SUFFIXES ]
           [SYNONYMS ]

SEQCK=NO        DIAGNOSTIC=NOLIST        DISPLAY=[KEYWORD ]
      YES                 LIST                  [SUFFIXES]
                                                [SYNONYMS]
```

a. Statement Identifier

STOPWORD= or DICTIONARY= must be first keyword, must begin in column 1.

b. table name

table name - must be less than 8 characters in length; must being with an alphabetic character and can not end with zero.

c. MODE identifier

CREATE, UPDATE or DISPLAY

d. Operation Keyword=word type

Required for CREATE/UPDATE modes; specifies whether keyword table is being added or deleted; and the type of keyword being processed.

```
{ADD=    }
{DELETE= }  operation

[STOPWORDS]
[KEYWORDS ]  word type
[SUFFIXES ]
[SYNONYMS ]
```

e. PAGE=nnk

One or two digits in range of 1-32K; default size is 1K

f. SEQCK=NO
YES

Specifies whether a sequence check is to be performed; default is NO.

g. DIAGNOSTIC=NOLIST
LIST

Specifies whether diagnostic reference listing is to be printed; default is NOLIST

h. DISPLAY=word type

Specifies the type of word lists that are to be printed in the DISPLAY mode; the default is all word types.

[KEYWORDS
SUFFIXES
SYNONYMS] word type

If two terms are selected, they must be enclosed in parentheses

e.g. DISPLAY=(KEYWORDS,SUFFIXES)

Value Statements

Value words are punched in free format in columns 1-71. Column 72 is ignored. Columns 73-80 may optionally contain a record sequence number. A word may not be split between two records. A suffix specification (explained below) may not be split between two records. A suffix specification need not appear in the same record as its root word. The parentheses which delimit suffix specifications and add-synonyms may appear anywhere including separate cards. Any number of blanks or commas may separate words, parentheses, and suffix specifications. All characters except the blank, comma, parenthesis, and equals symbol are considered to be part of a word. Words are stored in a table exactly as they appear in the input.

Stop Word Table Entries

Simple words are stored in stop word tables. Suffix and synonym notation does not apply.

Dictionary Entries

All words are stored in the dictionary exactly as they appear in the input. In addition, a changed form of the input word will be stored if a suffix specification follows the input word. Both forms of a word may be flagged as synonyms but their form is not changed.

Suffix Specification

This notation is used with words whose endings change when they are suffixed. Three such words are:

root

ARMY -ARMIES (changed the Y to I).
PLAN -PLANNED (add an N).
ARGUE-ARGUMENT (drop the E).

The specification consists of a 1-byte delete count (always a number) and zero or more characters that are to be added after deletion occurs. The specification (one or more separated by blanks or commas) must follow its root word and be enclosed in parentheses.

ARMY (1I) delete 1(Y), add I -- ARMI
PLAN (0N) delete none, add N -- PLANN
ARGUE (1) delete 1(E), add none -- ARGU

When a changed form dictionary word (ARMI, PLANN, ARGU) matches the significant characters of an argument word and when the remaining argument characters are a valid suffix, the root word (ARMY, PLAN, ARGUE) is substituted for the argument.

Synonyms

Synonomous relationships between words are defined by enclosing the related words in parentheses in an add-synonym function statement. Any number of words (two or more) may appear in one group. The same word may appear in any number of groups, which has the effect of combining the words in the common groups into a single group internally. UTNDXRMD collects all related words and selects the lowest in sequence to represent the group in the Index Data Set. This word is called the base word; all other words are called convert words. When an argument word matches any convert word, the base word is substituted for the argument.

If any of the words in an input group of synonyms is already a synonom in an existing table, the base word for the existing group will be used for the new group; i.e., all input words which are not synonyms will be stored in the dictionary as convert words in the existing group.

15.2 UTNDXKMD Output

For each file processed, UTNDXKMD lists all input statements with notes to indicate the action taken for the statements.

For each DISPLAY function, UTNDXKMD lists the contents of the table or dictionary in ascending alphanumeric sequence. It flags all convert words and suffixed words. It also shows all convert words as sublists with each base word so the user can see synonym groups.

15.3 UTNDXKMD Job Setup

The following JCL cards are used to invoke the cataloged procedure XKM which will maintain stop word tables and dictionaries.

```
//jobname  JOB  (standard parameters)
//stepname  EXEC XKM,LIB=libname
//XKM.SYSIN DD *
          (user-supplied control statements)
/*
```

where

libname is the name of the library containing stop word tables and dictionaries to be maintained.

Section 16

FORMAT DEFINITION TRANSLATOR UTILITY (UTODE)

UTODE is used to place format definitions on a user library. A format definition gives a description of a CRT display format and the Input Message Queue records to be created from data entered on the display. Before a user can call for the display format at a CRT terminal, the format definition must reside on the user library. UTODE creates skeleton and IMQ table control blocks from the format definition source statements and writes the control blocks into the user library using the display format name as the member name.

16.1 Input

Input to UTODE consists of one or more format definitions. The format definition source statements are described in Section 6 (FORMATTER) of the Terminal Processing Users Manual. The input source statements may be in punched cards or in card image records stored in a partitioned data set.

16.2 Job Setup

The following JCL statements illustrate the deck setup used to execute UTODE. In the first setup, the input definition source statements are in punched cards.

```
//AA      EXEC XUTODE,LIB=TEST360,VLIB='SER=MYPACK'
//SYSIN DD *

          Definition source statement cards.

/*
```

The following JCL would be used if the input were card image records stored in a library.

```
//AB       EXEC XUTODE,LIB=TEST360,VLIB='SER=MYPACK'
//SYSIN DD DSN=MYLIB(FORMAT1),VOL=SER=MYPACK2,
//          DISP=(SHR,KEEP),UNIT=2314,
//         DCB=(RECFB=FB,LRECL=80,BLKSIZE=800)
/*
```

DISTRIBUTION

CCTC CODES	COPIES
C124 (Reference and Record)	3
C124 (Record Copy) Stock	6
C240	20
C315	1
C341 (Maintenance Contractor)	10
C341 (Stock)	70

EXTERNAL

Director of Administrative Services, Office of the Joint Chiefs of Staff Attn: Chief, Personnel Division, Room 1A724, The Pentagon Washington, D.C. 20301	1
Director for Personnel, J-1, Office of the Joint Chiefs of Staff, Attn: Chief, Data Service Office, Room 1B738C, The Pentagon, Washington, D.C. 20301	1
Director for Operations, J-3, Office of the Joint Chiefs of Staff, Attn: P & AD, Room 2B870, The Pentagon, Washington, D.C. 20301	1
Director for Operations, J-3, Office of the Joint Chiefs of Staff, Attn: Deputy Director for Operations (Reconnaisance and Electronic Warfare) Room 2D921, The Pentagon, Washington, D.C. 20301	1
Director for Logistics, J-4, Office of the Joint Chiefs of Staff, Room 2E828, The Pentagon, Washington, D.C. 20301	1
Chief, Studies Analysis and Gaming Agency, Attn: Chief, Force Analysis Branch, Room 1D928A, The Pentagon, Washington, D.C. 20301	1
Automatic Data Processing, Liaison Office National Military Command Center, Room 2D901A, The Pentagon, Washington, D.C. 20301	1

EXTERNAL	COPIES
Automatic Data Processing Division Supreme Headquarters Allied Powers, Europe Attn: SA & P Branch, APO New York 09055------	1
Director, Defense Communications Agency, Office Of MEECN System Engineering, Attn: Code 960T, Washington, D.C. 20301----------	1
Director, Defense Communications Engineering Center, Hybrid Simulation Facility, 1860 Wiehl Avenue, Reston, VA 22070----------	1
Director, Defense Intelligence Agency Attn: DS - 5C2 Washington, D.C. 20301----------	5
Commander-in-Chief, Pacific, Attn: J6331, FPO San Francisco, 96610----------	1
Commander-in-Chief, US Army Europe and Seventh Army ATTN: OPS APO New York 09403---	1
Commanding General, US Army Forces Command, Attn: Data Support Division, Building 206, Fort McPherson, GA 30303----------	1
Commander, Fleet Intelligence Center, Europe, Box 18, Naval Air Station, Jacksonville, Florida 32212----------	1
Commanding Officer, Naval Air Engineering Center, Ground Support Equipment Department, SE 314, Building 76-1, Philadelphia, PA 19112	1
Commanding Officer, Naval Security Group Command, 3801 Nebraska Avenue, N.W. Attn: GP22, Washington, D.C. 20390----------	1
Commanding Officer, Navy Ships Parts Control Center, Attn: Code 712, Mechanicsburg, PA 17055	1
Headquarters, US Marine Corps, Attn: System Design and Programming Section (MC-JSMD-7) Washington, D.C. 20380----------	1

EXTERNAL	COPIES
Commanding Officer, US Army Forces Command Intelligence Center, Attn: AFIC-PD, Fort Bragg, NC 28307----------	1
Commander, US Army Foreign Science and Technology Center, Attn: AMXSJ-CS, 220 Seventh Street NE, Charlottsville, VA 22212--	1
Commanding Officer, US Army Security Agency, Command Data Systems Activity (CDSA) Arlington Hall Station, Arlington, VA 22212----------	1
Commanding Officer, US Army Security Agency Field Station - Augsburg, Attn: IAEADP, APO New York 09458----------	1
Commander, Fleet Intelligence Center, Atlantic, Attn: DPS, Norfolk, VA 23511----------	1
Commander, Fleet Intelligence Center, Pacific, Box 500, Pearl Harbor, HI 96860----------	1
Air Force Operations Center, Attn: Systems Division (XOOCSC) Washington, D.C. 20301-----	1
Commander, Armed Forces Air Intelligence Training Center, TTMNIM (360 FFS), Lowry AFB, Co 80230----------	1
Commander, Air Force Data Services Center, Attn: Director of System Support, Washington, D.C. 20330----------	1
Commander-in-Chief, US Air Forces in Europe, Attn: ACDI APO New York 09332----------	1
Commander, USAF Tactical Air Command, Langley AFB, VA 23665----------	1
Commander, Space and Missile Test Center, Attn: (ROCA) Building 7000, Vandenberg, AFB, CA 93437----------	1

EXTERNAL	COPIES
Naval Air Systems Command, Naval Air Station, Code 13999, Jacksonville, Florida 32212------	1
Commanding General, US Army Computer Systems Command, Attn: Support Operations Directorate, Fort Belvoir, VA----------	1
Defense Documentation Center, Cameron Station, Alexandria, VA 22314----------	12
TOTAL	159

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER CSM UM 15-78, VOLUME VII	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) NMCS Information Processing System 360 Formatted File System (NIPS 360 FFS) - Users Manual Vol VII - Utility Support (UT)		5. TYPE OF REPORT & PERIOD COVERED 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s)		8. CONTRACT OR GRANT NUMBER(s) DCA 100-77-C-0065
9. PERFORMING ORGANIZATION NAME AND ADDRESS International Business Machines, Corp. Rosslyn, Virginia		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS National Military Command System Support Center The Pentagon, Washington, D.C. 20301		12. REPORT DATE 1 September 1978 13. NUMBER OF PAGES 94
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)
Copies of this document may be obtained from the Defense Documentation Center, Cameron Station, Alexandria, Virginia 22314.
This document has been approved for public release and sale; its distribution is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)
This volume defines the capabilities of NIPS 360 FFS Utility (UT) components. It describes the function of each utility, its inputs, its outputs, and serves as a reference for the knowledgeable user of these components.

This document supersedes CSM UM 15-74, Volume VII.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)